

Association Rules in the Relational Calculus

Oliver Schulte, Flavia Moser, Martin Ester and Zhiyong Lu
 School of Computing Science
 Simon Fraser University
 Burnaby, B.C., Canada
 {oschulte,fmoser,ester,zhiyongl}@cs.sfu.ca

February 2, 2008

Abstract

One of the most utilized data mining tasks is the search for association rules. Association rules represent significant relationships between items in transactions. We extend the concept of association rule to represent a much broader class of associations, which we refer to as *entity-relationship rules*. Semantically, entity-relationship rules express associations between properties of related objects. Syntactically, these rules are based on a broad subclass of safe domain relational calculus queries. We propose a new definition of support and confidence for entity-relationship rules and for the frequency of entity-relationship queries. We prove that the definition of frequency satisfies standard probability axioms and the Apriori property.

1 Introduction

One of the goals of data mining is to discover interesting relationships from data. Association rules express relationships that hold with sufficient frequency but not always. For example, it may be the case that not all managers earn over \$60,000 a year, but that 90% of managers do. The logical form of an association rule is that of an implication $p \rightarrow q$ where p and q hold together sufficiently often (the “support” of the rule) and q holds sufficiently often given that p holds (the “confidence” of the rule). The traditional concept of association rules severely limits the complexity of the expressions p and q and thereby limits the class of relationships a data miner can capture. Essentially, p and q may be only simple conjunctions, like an itemset. Thus we cannot have rules based on Boolean combinations, such as negations or nested combinations. An example of a relationship involving a negation would be a negative factor, such as “students who have not taken an introductory database course do poorly in datamining courses”. An example of a nested Boolean combination would be “students who are math majors or computer science majors, and who have done well in a discrete mathematics course or in an algorithms course, do well in complexity theory”. Another class of relationships that association rules cannot express involves quantification and relating objects to each other. An example would be the rule “residents who have a neighbour with high incomes tend to have a high income themselves”.

The goal of this paper is to extend the concept of an association rule to a large class of expressions that we refer to as *entity-relationship queries* (ER queries). Intuitively, entity-relationship queries express dependencies among entities and their properties. Entity-relationship queries are a large subclass of the safe queries. Safe queries correspond to an expressive subset of first-order logic that allows for nested Boolean expressions and quantification. We provide a definition of the frequency of an entity-relationship query. This extends the notion of an association rule to implications of the form $p \rightarrow q$ where $p \wedge q$ is an ER query; we refer to rules of this form as *entity-relationship rules*. From our definition of the frequency of an ER query we immediately obtain a definition of the support of an ER rule, namely the frequency $fr(p \wedge q)$.

TV-Program(<u>Prog-Name</u> :string)
TV-Station(<u>Station-Name</u> :string, Area:integer)
WeekdayTV(TV-Program:string,TV-Station:string,Viewers:integer,Sponsor:string)
WeekendTV(TV-Program:string,TV-Station:string,Viewers:integer,Sponsor:string)

Table 1: A relational schema for a TV survey model. Key fields are underlined. The schema lists TV programs and stations, and records for each combination of weekday program and station, how many viewers view the program on that station, and who sponsors the program. The same information is recorded for weekend programs.

Our definition of frequency for ER queries generalizes previous work on defining association rules in a multi-relational setting. [1] discusses extending itemset rules with negations and motivates the usefulness of this extension. The query extension approach of the WARMR system [4] presents a special class of entity-relationship rules that allows conjunctions of nonnegated statements and existential quantification. Our concept of ER rules features in addition negations, universal quantification, nested quantifiers, and nested Boolean combinations. Thus one contribution of this paper is an extended rule format. A characteristic that distinguishes our approach from previous work is that previous approaches assume a given target table that defines a base set of tuples for evaluating the support of a query. In contrast, we start with a query and define a natural base set of tuples for evaluating the support of the query. We can think of this approach as dynamically generating entity sets for a given query rather than evaluating queries with respect to a fixed entity set. Thus the second main contribution of this paper is a new definition of support for rules in our extended format.

The paper is organized as follows. First we review basic relational database concepts such as the relational schema and the domain relational calculus. Then we introduce the concept of an entity query and define the frequency of a query in this class of queries. This definition provides the basis for the notion of an entity-relationship rule and for defining the support of an entity-relationship rule. We compare entity-relationship queries to frequent itemsets and to the rule language of the WARMR system. The final section establishes several important formal properties of query frequencies as we define them and shows that they satisfy the Apriori property, that is, the frequency of a conjunction is no greater than the frequency of its conjuncts.

2 Entities in the Domain Relational Calculus

This section presents standard background material from database theory. The first subsection reviews relational schemas, and introduces the new concept of an *entity field*. Semantically, entity fields are those that store values (constants) that refer to entities. The second subsection defines the standard notion of a safe query in the domain relational calculus, and the third introduces a subclass of safe queries that we term *entity-relationship queries*.

2.1 Entities in Relational Schemas

We begin with a standard **relational schema** containing a set of tables, each with key fields, descriptive attributes, and possibly foreign key pointers. We use the notation T to refer to a generic table that may represent either an entity set or a relationship set, and for an index we use T^i . A field named *name* in table T is denoted by $T.name$. Table 1 shows a relational schema for a TV survey database; this example is adapted from [6, Sec.2]. Tables 2–4 display *relation instances* for the TV survey schema.

We assume that the tables in the relational schema can be divided into *entity tables* and *relationship tables*. This is the case whenever a relational schema is derived from an entity-relationship model (ER model) [8, Ch.2.2]. Intuitively, an entity table corresponds to a type of entity, and a relationship table represents a relation between entity types. In our TV survey example, there are two types of entities: TV programs represented in the TV-Program table, and TV stations represented in the TV-Station table. We

<u>TV-Program</u>	<u>TV-Station</u>	Viewers	Sponsor
Gilmore	Global	10	Avon
Gilmore	CBS	12	La Senza
Hockey Night	CBC	20	RBC

Table 2: Television Survey: Weekday TV.

<u>TV-Program</u>	<u>TV-Station</u>	Viewers	Sponsor
Gilmore	Global	8	Avon
Hockey Night	CBC	14	Schwab
Simpsons	CBS	10	RBC
Daily Show	CBC	6	La Senza

Table 3: Television Survey: Weekend TV.

now introduce two assumptions concerning the relational schema that facilitate the definition of entity-relationship queries and their frequencies.

Unary Key Assumption We assume that every entity table has a single key field.

The advantage of the unary key assumption is that given this assumption, a single key field in the relational schema refers to a single entity. The assumption holds in our TV survey schema because the two entity tables have key fields TV-Program.Prog-Name and TV-Station.Station-Name respectively. Although it is not always natural to define entities with a single key field, there is no loss of generality because we can always form a single composite key field from a list of key fields. For example, if in a Professor table there are two key fields FirstName, LastName, we can form a composite key field $\langle \text{FirstName}, \text{LastName} \rangle$. Our second assumption is the following.

Global Name Assumption We assume that for every entity e , there is a unique constant c such that in every table, the constant c denotes entity e .

The global name assumption is important because it allows us to recognize when the same entity occurs in different tables. In the AI literature, a similar assumption is often referred to as the “unique name assumption” [7, Ch.14]. The assumption does not amount to a loss of generality because if the same constant c is used in different tables to refer to different entities, we can simply index c to distinguish these occurrences. For example, if we have two different transaction tables Transaction1 and Transaction 2, and there is a transaction 1 in both, we could change the entry in the first table to refer to 1-1 and in the second table to refer to 1-2. A natural alternative to indexing constants would be to adopt a convention to the effect that a key field $T.key$ in table T refers to different entities than key field $T'.key$ in table T' if and only if the names of the key fields in the two tables are different. For example, if we have a table for Employees and another for Managers, labelling the key field in each table as “ssn” indicates that a given social security number refers to the same person no matter where it appears. In contrast, labelling the key field in the Transactions1 table “T1-number” and the key field in the Transactions2 table as “T2-number” indicates that the transaction numbers in different tables refer to different transactions.

<u>Station-Name</u>	Area
Global	1
CBS	2
CBC	3

Table 4: Television Survey: Stations and Areas.

Symbol Type	Notation	Comment
Constants	c_1, c_2, \dots	At most countably many constants
Predicate Symbols	P_1, P_2, \dots, P_k	Exactly one predicate for each table T^i
Logical Symbols	$\exists, \forall, \wedge, \vee, \neg$	
Comparison Operators	$=, <, >, \leq, \geq, \neq$	

Table 5: The Basic Vocabulary of our DRC language for a given database schema \mathcal{D} with tables T^1, \dots, T^k .

In many applications, the global name assumption is enforced through foreign key constraints. To illustrate, in the TV example, we may suppose that the field WeekdayTV.TV-Station is a foreign key pointer to the field TV-Station.Station-Name, and that the field WeekendTV.TV-Station is a foreign key pointer to the same field. So the string constant “CBS” refers to the CBS network represented in the TV-Station table, whether “CBS” appears in an instance of the WeekdayTV relation or in an instance of the WeekendTV relation.

Given the unary key and global name assumptions, the following is a valid definition of how tables, key fields and constants are associated with entities.

Definition 1 *Let \mathcal{D} be a database instance.*

1. An **entity table** is a table T with a single key field.
2. A field is an **entity field** if (1) the field is the key of an entity table, or (2) the field is a foreign pointer to the key of an entity table.
3. A constant c is an **entity constant** if c appears in an entity field.

Examples. Let \mathcal{D} be the TV survey database instance from Tables 2–4. The entity keys are TV-Program.Prog-Name, TV-Station.Station-Name, WeekdayTV.TV-Program, WeekdayTV.TV-Station, WeekendTV.TV-Program, WeekendTV.TV-Station. Entity constants include “CBS” and “Simpsons”.

Next we review the domain relational calculus, which is a logical query language based on a given relational schema.

2.2 Safe Queries in the Domain Relational Calculus

We first define the formal language of the domain relational calculus, including the well-formed formulas of the calculus. Then we define an important subclass of formulas known as safe queries. Our presentation follows the standard approach, see for example [8, Ch.3].

2.2.1 The Formal Language of the Domain Relational Calculus

In the domain relational calculus (DRC), for every table T^i in the database schema there is exactly one predicate P_i in the logical language. The number of fields in the table T^i is the arity of the predicate P_i . If T^i is an entity table, then P_i is an **entity predicate**. By the unary key assumption, an entity table T^i has a single key field; we adopt the convention that the key field is the first argument in the entity predicate P_i . The complete logical vocabulary of the DRC is listed in Table 5.

Example. In the TV survey model, we have the predicates shown in Table 6.

Thus we may write $WeekdayTV(\text{“GilmoreGirls”}, \text{“CBS”}, 12, \text{“La Senza”})$ to assert that “Gilmore Girls” is shown on “CBS” on weekdays, with 12,000 viewers, and sponsored by La Senza. The notion of a well-formed formula is the usual one for this vocabulary.

Definition 2 *Well-Formed Formulas of the Domain Relational Calculus*

1. A constant c or variable X is a term.

Table 6: Predicates of our Logical Query Language for the TV survey model.

<i>Predicates</i>	Arity
TV-Program(PN)	1
TV-Station(SN,A)	2
WeekdayTV(PN,SN,V,S)	4
WeekendTV(PN,SN,V,S)	4

Table 7: Examples of Valid Expressions for the database schema for the TV survey.

Expression	Type
$V \geq 10$	atomic formula with V free
$\exists S. \exists SN. \exists V. \text{WeekdayTV}(P, SN, V, S)$	quantified formula with P free
$\exists S. \exists SN. \exists V. \text{WeekdayTV}(P, SN, V, S) \wedge V \geq 10 \wedge$ $\exists S. \exists SN. \exists V. \text{WeekendTV}(P, SN, V, S) \wedge V \geq 10$	conjunction of quantified formulas

2. If P is a predicate symbol of arity k and t_1, \dots, t_k are k terms, then $P(t_1, \dots, t_k)$ is an atomic formula.
3. If t, t' are two terms, then a comparison $t_1 \theta t_2$ is an atomic formula.
4. If F is a formula and X is a variable, then $\neg F, \exists X.F, \forall X.F$ are formulas.
5. If F_1 and F_2 are formulas, then so are $F_1 \wedge F_2$ and $F_1 \vee F_2$.
6. All formulas are formed by the repeated application of the previous rules.

Examples. Table 7 gives examples of valid expressions and their types pertaining to the TV survey.

We next define the result or output of a DRC query. The first step is to define what ground formulas are satisfied in a database instance \mathcal{D} ; a formula is **ground** if it contains no variables. The second step is to define which closed queries F with no free variables are satisfied in a database instance \mathcal{D} ; as usual in logic, we write $\mathcal{D} \models F$. Let $F[X_1/t_1, \dots, X_k/t_k]$ be the formula that results from replacing all free occurrences of each X_i in F with the term t_i .

1. If t, t' are two constants, then $\mathcal{D} \models t\theta t'$ iff $t\theta t'$ holds.
2. $\mathcal{D} \models P_i(c_1, \dots, c_k)$ iff $\langle c_1, \dots, c_k \rangle$ is a tuple in table T^i .
3. $\mathcal{D} \models F_1 \vee F_2$ iff $\mathcal{D} \models F_1$ or $\mathcal{D} \models F_2$; similarly $\mathcal{D} \models F_1 \wedge F_2$ iff $\mathcal{D} \models F_1$ and $\mathcal{D} \models F_2$; and $\mathcal{D} \models \neg F_1$ iff $\mathcal{D} \not\models F_1$.
4. $\mathcal{D} \models \exists X.F$ iff there is a constant c in the DRC language such that $\mathcal{D} \models F[X/c]$; similarly $\mathcal{D} \models \forall X.F$ iff for all constants c we have $\mathcal{D} \models F[X/c]$.

Let $F(X_1, \dots, X_m)$ be a query with free variables X_1, \dots, X_m . Then on database instance \mathcal{D} the query F returns the set of all tuples that make F true when substituted in F . Formally, we write

$$\text{tuples}_{\mathcal{D}}(F) \equiv \{ \langle c_1, \dots, c_m \rangle : \mathcal{D} \models F[X_1/c_1, \dots, X_m/c_m] \}.$$

This definition assumes that the constants in the language include all constants that appear in the database tables, which involves no loss of generality.

Examples. Let \mathcal{D} be the database instance from Tables 2–4. Table 8 shows the results of our example queries for this database instance.

Query Formula F	Result $tuples_{\mathcal{D}}(F)$
$F_1 = \exists S. \exists SN. \exists V. WeekdayTV(P, SN, V, S) \wedge V \geq 10$	{“Gilmore”, “Hockey Night”}
$F_2 = \exists S. \exists SN. \exists V. WeekendTV(P, SN, V, S) \wedge V \geq 10$	{“Hockey Night”, “Simpsons”}
$F_1 \wedge F_2$	{“Hockey Night”}

Table 8: Results of Query Formulas on the database instance \mathcal{D} from Tables 2–4.

Query Formula F	Safe?
$F_1 = \exists S. \exists SN. \exists V. WeekdayTV(P, SN, V, S) \wedge V \geq 10$	yes
$F_2 = \exists S. \exists SN. \exists V. WeekendTV(P, SN, V, S) \wedge V \geq 10$	yes
$F_1 \wedge F_2$	yes
$F_1 \vee F_2$	yes
$\neg F_1$	no
$\neg F_1 \vee F_2$	no
$F_1 \wedge \neg F_2$	yes

Table 9: Examples of safe and unsafe queries for the TV survey database schema of Table 1.

2.2.2 Safe Queries

It is customary to restrict the set of formulas that may serve as free variables in a query (“query variables” for short) to ensure that the result set of tuples satisfying query formulas are bounded and “domain-independent” [8, Ch.3.8]. To this end we adopt the notion of a safe query. The intuition behind this concept is that the results of safe queries should be restricted to selection conditions applied to (combinations of) tables in the database. For example, the query $\neg TVProgram(X)$ with free variable X is not safe because the range of constants satisfying this query is not bound by any table in the database. The key idea in the definition of safe query is to conjoin a query formula F to a restriction of the form $P \wedge F$ where P is a basic predicate in the language and hence refers to a table in the database. As is well-known, the expressive power of safe queries is exactly equivalent to that of relational algebra [2]. Safe queries are formally defined as follows [8, Ch.3.8].

1. Replace the $\forall X$ quantifier by $\neg \exists X \neg$.
2. Whenever \vee is used to connect $F_1 \vee F_2$, the two formulas have the same set of free variables.
3. Consider any maximal subformula consisting of the conjunction of one or more formulas $F_1 \wedge \dots \wedge F_m$. Then all variables X appearing free in any of the F_i must be limited as follows. The variable X must be free in some *non-negated* F_i satisfying *one* of the following conditions.
 - (a) F_i is not a comparison.
 - (b) F_i is $X = c$ where c is a constant.
 - (c) F_i is $X = Y$, and Y is limited.
4. A \neg operator may apply only to a formula in a conjunction of the type discussed in the previous rule.

Examples. Table 9 gives examples of safe and unsafe queries for the TV database schema from Table 1.

This completes our review of basic concepts from relational database theory. We now come to the restriction of safe queries to entity-relationship queries.

2.3 Definition of Entity-Relationship Queries

The basic idea behind our definition of an ER query is that free variables should be limited in such a way as to guarantee that they must refer to entities. Intuitively, an ER query is one whose free variables refer to entities. The precise definition is as follows.

Definition 3 Let \mathcal{D} be a database instance.

1. A variable X is an **entity variable candidate** for a DRC formula F if
 - (a) X is not quantified over in any part of F
 - (b) if an expression $X\theta t$ appears in F , then θ is $=$ or \neq , and if t is a constant c , then c is an entity constant in \mathcal{D} .
 - (c) if an expression $P(_, X, _)$ appears in F , then the argument position of X in $P(_, X, _)$ is an entity field.
2. A variable X is an **entity variable** for F if
 - (a) X is an entity variable candidate for F , and
 - (b) if an expression $X = Y$ or $X \neq Y$ appears in F , then Y is an entity variable candidate.
3. An **entity-relationship (ER) query** F for database instance \mathcal{D} is a safe DRC query such that all the free variables in F are entity variables for F given \mathcal{D} .

Examples. Let \mathcal{D} be the TV survey database instance from Tables 2–4. In the formula

$$\exists S. \exists SN. \exists V. \text{WeekdayTV}(P, SN, V, S) \wedge V \geq 10$$

the variable P is an entity variable, and so the formula is an ER query. The formula

$$\exists S. \exists SN. \text{WeekdayTV}(\text{“Gilmore Girls”}, SN, V, S)$$

is safe but not an ER query because the free variable V is not an entity variable.

3 The Frequency of Entity-Relationship Queries

Our basic idea is that the limiting conditions in safe queries specify the domain from which values for a free variable X are to be drawn. Once the domain for the free variables is defined for a given formula F , we can take the frequency of the formula F to be the number of assignments to the free variables that satisfy the formula divided by the size of the domain for the formula. Safe queries are a natural class of queries for this approach because these queries specify the range from which result tuples may be drawn by restricting these results to subsets of tables in the database (cf. Section 2.2.2).

The main issue in our definition concerns the correct domain for conjunctions or intersections. For a simple example, consider a database schema with two entity tables Professor and Customer. The query $\text{Professor}(X) \wedge \text{Customer}(X)$ returns entities that are both professors and customers. What should be the base domain for this query? If there are many more customers than professors, we may get quite different frequency counts if we take the base domain to be Professor than if we take it to be Customer. So neither of these seems the right choice. Intuitively the base domain should be a symmetric function of the two classes mentioned in the query. The two natural symmetric set-theoretic operations are intersection and union. If we take the intersection as the base domain, the frequency of conjunctions without further selection conditions is always 100%, which does not seem right. In particular for our ultimate goal of defining the support of association rules, this is unsatisfactory. Our proposal is therefore to use the *union* of the two entity sets involved in the conjunction. Another way to look at the union is that it represents a kind of closed world assumption: If Professors and Customers are the only entity types mentioned in the selection conditions of the query, then the members of these entity types are exactly the potential answers to the query.

The closed world assumption is also the basis for our frequency definition for queries with negation. For example, consider a safe query such as $\text{Professor}(X) \wedge \neg \text{Customer}(X)$. Since Professors and Customers are the only entity types mentioned in this query, we take the base domain again to be the union of these two

Query Formula F , Reference Domain $dom_{\mathcal{D}}(F, X)$
$F_1 = \exists S. \exists SN. \exists V. WeekdayTV(P, SN, V, S) \wedge V \geq 10$ $dom_{\mathcal{D}}(F_1, X) = \{\text{"Gilmore"}, \text{"Hockey Night"}\}$
$F_2 = \exists S. \exists SN. \exists V. WeekendTV(P, SN, V, S) \wedge V \geq 10$ $dom_{\mathcal{D}}(F_2, X) = \{\text{"Gilmore"}, \text{"Hockey Night"}, \text{"Simpsons"}, \text{"Daily Show"}\}$
$F_3 = F_1 \wedge F_2$ $dom_{\mathcal{D}}(F_3, X) = \{\text{"Gilmore"}, \text{"Hockey Night"}, \text{"Simpsons"}, \text{"Daily Show"}\}$
$F_4 = F_1 \vee F_2$ $dom_{\mathcal{D}}(F_4, X) = \{\text{"Gilmore"}, \text{"Hockey Night"}, \text{"Simpsons"}, \text{"Daily Show"}\}$
$F_5 = F_1 \wedge \neg F_2$ $dom_{\mathcal{D}}(F_5, X) = \{\text{"Gilmore"}, \text{"Hockey Night"}, \text{"Simpsons"}, \text{"Daily Show"}\}$

Table 10: Reference Domains for various formulas in the TV survey database instance \mathcal{D} from Tables 2–4.

sets. The fact that Professors are mentioned positively and Customers negatively does not make a difference to the base domain, but it does make a difference to the result of the query and hence to its frequency.

On the basis of this proposal, we can now recursively assign a domain to an entity variable in a formula F given a database instance \mathcal{D} . We begin with just one free query variable and then tackle the more complicated case of queries with more than one free variable.

3.1 Definition of Frequency for Queries With One Free Query Variable

We denote the base domain of an entity variable X in a query F relative to a database instance \mathcal{D} as $dom_{\mathcal{D}}(F, X)$. As we think of variable X as referring to the domain $dom_{\mathcal{D}}(F, X)$, we term $dom_{\mathcal{D}}(F, X)$ the **reference domain** of X in the context of query F .

Definition 4 Let \mathcal{D} be a database instance with ER formula F .

1. If F is $P_i(t_1, \dots, t_k)$, and X occurs in F , then $dom_{\mathcal{D}}(F, X) = \pi_X[tuples_{\mathcal{D}}(F)]$. If X is not a free variable in F , then $dom_{\mathcal{D}}(F, X) = \emptyset$. Here we think of $tuples_{\mathcal{D}}(F)$ as a relation whose columns correspond to the free variables of F . For example, the query $p(X, Y, Z)$ returns a relation with triples, and we can think of the first column as named X and the second as named Y . The expression π refers to the projection operator of relational algebra (with elimination of duplicates).
2. Let F be a single atomic comparison of the form $Y \theta t$ where t is either a variable or a constant. If F is $X = c$, then $dom_{\mathcal{D}}(F, X) = \{c\}$. Otherwise $dom_{\mathcal{D}}(F, X) = \emptyset$.
3. If F is $\neg G$ for some formula G , then $dom_{\mathcal{D}}(F, X) = dom_{\mathcal{D}}(G, X)$.
4. If F is $F_1 \vee F_2$ or $F_1 \wedge F_2$, then $dom_{\mathcal{D}}(F, X) = dom_{\mathcal{D}}(F_1, X) \cup dom_{\mathcal{D}}(F_2, X)$.
5. If F is $\exists Y. G$, where $Y \neq X$, then $dom_{\mathcal{D}}(F, X) = dom_{\mathcal{D}}(G, X)$. If F is $\exists X. G$, then $dom_{\mathcal{D}}(F, X) = \emptyset$.

Examples. Let \mathcal{D} be the TV survey database instance from Tables 2–4. Table 10 gives examples of reference domains for various ER queries.

As this definition shows, we think of basic predicates as specifying the range from which entities are drawn. Conditions of the form $p(t_1, \dots, X, \dots, t_k)$ or $X = c$ we view as “direct bounds” that determine the reference domain of X . Variable equations of the form $X = Y$ we view as “selection conditions” that are applied after an entity has been specified. These do not affect the reference domain of X but only the result of the query. Another type of selection are restrictions on descriptive attributes, such as $V \geq 10$ in the queries in Table 10.

Now the frequency of an ER query is defined as follows.

Query Formula F	Frequency $fr_{\mathcal{D}}(F)$
$F_1 = \exists S. \exists SN. \exists V. \text{WeekdayTV}(P, SN, V, S) \wedge V \geq 10$	1
$F_2 = \exists S. \exists SN. \exists V. \text{WeekendTV}(P, SN, V, S) \wedge V \geq 10$	1/2
$F_1 \wedge F_2$	1/4
$F_1 \vee F_2$	3/4
$F_1 \wedge \neg F_2$	1/4

Table 11: Frequencies for various formulas in the TV survey database instance \mathcal{D} from Tables 2–4.

Definition 5 Let F be an ER query with free variable X such that $dom_{\mathcal{D}}(F, X) \neq \emptyset$. Then

$$fr_{\mathcal{D}}(F) \equiv \frac{|tuples_{\mathcal{D}}(F)|}{|dom_{\mathcal{D}}(F, X)|}.$$

In Section 5 we establish several formal properties of the frequency of a query according to this definition, for example that the frequency is a number between 0 and 1.

Examples. Let \mathcal{D} be the TV survey database instance from Tables 2–4. Table 11 illustrates the frequencies of various queries.

3.2 Definition of Frequency of Queries With More Than One Free Variable

We assign a domain to every tuple of entity variables in a formula F given a database instance \mathcal{D} , which we denote as $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\})$. Our basic idea is to consider a result tuple $\langle c_1, \dots, c_m \rangle$ as denoting a *composite entity* formed by combining m single entities. For example, consider the rule $Neighbour(X, Y) \wedge (\exists I. Income(X, I) \wedge I > \$100,000) \rightarrow (\exists I. Income(Y, I) \wedge I > \$100,000)$. (The symbol \rightarrow does not denote logical implication but defines an association rule; see Section 4.) This says that if X has an income over \$100,000, then it is likely that a neighbour Y of X also has an income of \$100,000. The support of this rule is the frequency of the query $Neighbour(X, Y) \wedge (\exists I. Income(X, I) \wedge I > \$100,000) \wedge (\exists I. Income(Y, I) \wedge I > \$100,000)$. This query has two free variables X and Y . The reference domain comprises the entries in the *Neighbour* table, that is, the pairs $\langle X, Y \rangle$ in the table. Other examples of natural composite entities include relations like reservations or purchases. The idea of treating tuples in a relational table as composite “individuals” is familiar in the propositionalization literature [5, 3] (for example, chemical molecules may be treated as single entities although molecules are composed of different elements that are also represented in the relational schema). Applying this idea requires a further constraint on ER queries: the free variables $\{X_1, \dots, X_m\}$ must be “bound together” in a limiting condition rather than separately. For example, the query $P(X) \wedge X = Y$ is a safe ER query but the answer pairs $\langle x, y \rangle$ are not bound to the key fields of any tuple; an example of the same character is the query $P(X) \wedge Q(Y)$. To rule out such cases, we impose the following condition.

Definition 6 A *literal* is an atomic formula or its negation. An ER query F is **valid for variables** X_1, \dots, X_m if for every maximal conjunction $L = L_1 \wedge \dots \wedge L_k$ consisting only of literals, L contains a conjunction of the form $X_1 = c_1 \wedge \dots \wedge X_m = c_m$, or L contains a conjunct $P(t_1, \dots, t_k)$ where all variables $\{X_1, \dots, X_m\}$ occur in $P(t_1, \dots, t_k)$. An ER query F is **valid** if F is valid for the set of its free variables.

Examples follow below in this section. In the case with only one free query variable X , the definition of safe query implies that every entity query is valid. Now let us consider the definition of a reference domain for valid ER queries with one or more free variables. As in the case with just one query variable, we term $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\})$ the **reference domain** of $\{X_1, \dots, X_m\}$ in the context of query F . Consider the basic case of an atomic formula $F = P(t_1, \dots, t_m)$ first. In keeping with the idea behind safe queries, we can think of such formulas as specifying a basic range for the result tuples in a query. So suppose that the free variables in the atomic formula are Y_1, Y_2, \dots, Y_k . If our query variables X_1, \dots, X_m are not *all* contained in the set $\{Y_1, Y_2, \dots, Y_k\}$, we consider that the “composite key” X_1, \dots, X_m does not appear in the query,

Query Formula F , Reference Domain $dom_{\mathcal{D}}(F, X)$
$F_1 = \exists S. \exists V. \text{WeekdayTV}(P, SN, V, S) \wedge V > 10$ $dom_{\mathcal{D}}(F_1, X) = \{\langle \text{"Gilm."}, \text{"Glo."} \rangle, \langle \text{"Gilm."}, \text{"CBS"} \rangle, \langle \text{"Hock. N."}, \text{"CBC"} \rangle\}$
$F_2 = \exists S. \exists V. \text{WeekendTV}(P, SN, V, S) \wedge V > 10$ $dom_{\mathcal{D}}(F_2, X) = \{\langle \text{"Gilm."}, \text{"Glo."} \rangle, \langle \text{"Hock. N."}, \text{"CBC"} \rangle, \langle \text{"Simps."}, \text{"CBS"} \rangle, \langle \text{"Daily Sh."}, \text{"CBC"} \rangle\}$
$F_3 = F_1 \wedge F_2$ $dom_{\mathcal{D}}(F_3, X) = \{\langle \text{"Gilm."}, \text{"Glo."} \rangle, \langle \text{"Gilm."}, \text{"CBS"} \rangle, \langle \text{"Hock. N."}, \text{"CBC"} \rangle, \langle \text{"Simps."}, \text{"CBS"} \rangle, \langle \text{"Daily Sh."}, \text{"CBC"} \rangle\}$

Table 12: Reference Domains for various formulas in the TV survey database instance \mathcal{D} from Tables 2–4. The free variables query variables are P and SN , corresponding to pairs of programs-stations.

and $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \emptyset$. Otherwise we consider the query result $tuples_{\mathcal{D}}(F)$ as a relation with k columns, of which m are named X_1, \dots, X_m . For example, the query $p(X, Y, Z)$ returns a relation with triples, and we can think of the first column as named X and the second as named Y . Thus we can take $\pi_{\langle X_1, \dots, X_m \rangle} tuples_{\mathcal{D}}(F)$ to be the reference domain of the entity variables X_1, X_2, \dots, X_m in the query F . This leads to the following inductive definition. The main difference with the definition for a single query variable is that we need to treat conjunctions like $X_1 = c_1 \wedge \dots \wedge X_m = c_m$ as a single compound statement.

Definition 7 Let \mathcal{D} be a database instance with ER formula F and let X_1, \dots, X_m be a list of variables.

1. If F is $P(t_1, \dots, t_k)$, and all variables X_1, \dots, X_m occur in $P(t_1, \dots, t_k)$, then $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \pi_{\langle X_1, \dots, X_m \rangle} tuples_{\mathcal{D}}(F)$, where π is the projection operation of relational algebra. Otherwise $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \emptyset$.
2. Let F be a single atomic comparison of the form $Y \theta t$ where t is either a variable or a constant.
 - (a) Suppose that $m = 1, Y = X_1$ and the comparison is $X_1 = c$ (i.e., we just have a single free variable X_1 and the atomic formula requires X_1 to be equal to a constant c .) In that case $dom_{\mathcal{D}}(F, \{X_1\}) = \{c\}$.
 - (b) Otherwise $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \emptyset$.
3. Let F be a maximal conjunction of $k > 1$ formulas, such that $F = C_1 \wedge \dots \wedge C_k$.
 - (a) If F is a conjunction of the form $C \wedge X_1 = c_1 \dots \wedge X_m = c_m$, then $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = dom_{\mathcal{D}}(C, \{X_1, \dots, X_m\}) \cup \{c_1, \dots, c_m\}$.
 - (b) Otherwise $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \bigcup_{i=1}^k dom_{\mathcal{D}}(C_i, \{X_1, \dots, X_m\})$.
4. If F is $F_1 \vee F_2$, then $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = dom_{\mathcal{D}}(F_1, \{X_1, \dots, X_m\}) \cup dom_{\mathcal{D}}(F_2, \{X_1, \dots, X_m\})$.
5. If F is $\neg G$ for some formula G , then $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = dom_{\mathcal{D}}(G, \{X_1, \dots, X_m\})$.
6. If F is $\exists Y. G$, where $Y \notin \{X_1, \dots, X_m\}$, then $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = dom_{\mathcal{D}}(G, \{X_1, \dots, X_m\})$. If F is $\exists X_i. G$ for some $X_i \in \{X_1, \dots, X_m\}$, then $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \emptyset$.

It is easy to check that this definition agrees with Definition 4 for queries with just one free variable.

Examples. Consider the query “find all program-station pairs that achieve a viewership of over 10,000 on both weekdays and weekends”. In the domain relational calculus, this query may be formulated as $[\exists S. \exists V. \text{WeekdayTV}(P, SN, V, S) \wedge V > 10] \wedge [\exists S. \exists V. \text{WeekendTV}(P, SN, V, S) \wedge V > 10]$. Table 12 shows the calculation of the reference domain for this formula on the database instance of Tables 2–4.

Now the frequency of an ER query is defined as follows.

Query Formula F	Frequency $fr_{\mathcal{D}}(F)$
$F_1 = \exists S. \exists V. \text{WeekdayTV}(P, SN, V, S) \wedge V > 10$	$2/3$
$F_2 = \exists S. \exists V. \text{WeekendTV}(P, SN, V, S) \wedge V > 10$	$1/4$
$F_1 \wedge F_2$	$1/5$
$F_1 \vee F_2$	$2/5$
$F_1 \wedge \neg F_2$	$1/5$

Table 13: Frequencies for various formulas in the TV survey database instance \mathcal{D} from Tables 2–4. The free variables query variables are P and SN , corresponding to pairs of programs-stations.

Definition 8 Let F be an ER query whose free variables are X_1, \dots, X_m where $dom_{\mathcal{D}}(\{X_1, \dots, X_m\}, F) \neq \emptyset$. Then

$$fr_{\mathcal{D}}(F) \equiv \frac{|tuples_{\mathcal{D}}(F)|}{|dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\})|}.$$

Table 13 illustrates the frequencies of various queries.

4 Entity-Relationship Rules

We finally obtain the notion of an ER association rule, or ER rule for short.

4.1 Definition of Confidence and Support for ER rules

Given the concepts we have developed so far, the definition of confidence and support for an entity-relationship rule are straightforward.

Definition 9 Let \mathcal{D} be a database instance.

1. An **ER association rule** is an implication of the form $F \rightarrow G$, where the free variables of G are the same as or contained in the free variables of F , and $F \wedge G$ is a valid ER query.
2. The **confidence** of an ER association rule $F \rightarrow G$ is given by

$$con_{\mathcal{D}}(F \rightarrow G) \equiv \frac{|tuples_{\mathcal{D}}(F \wedge G)|}{|tuples_{\mathcal{D}}(F)|}.$$

3. The **support** of an ER association rule $F \rightarrow G$ is given by

$$support_{\mathcal{D}}(F \rightarrow G) \equiv fr_{\mathcal{D}}(F \wedge G).$$

As usual with association rules, the implication $F \rightarrow G$ does not indicate logical implication (whenever F is true, so is G) but instead denotes a probabilistic relationship.

Example. Let \mathcal{D} be the TV survey database instance from Tables 2–4. Let F_1 be the formula

$$\exists S. \exists SN. \exists V. \text{WeekdayTV}(P, SN, V, S) \wedge V \geq 10$$

and let F_2 be the formula

$$\exists S. \exists SN. \exists V. \text{WeekendTV}(P, SN, V, S) \wedge V \geq 10.$$

Consider the rule $F_1 \rightarrow F_2$. The support of this rule is $fr_{\mathcal{D}}(F_1 \wedge F_2) = 1/4$ (see Table 11). The confidence is

$$\frac{|\{ \text{"Gilmore"}, \text{"Hockey Night"} \} \cap \{ \text{"Hockey Night"}, \text{"Simpsons"} \}|}{|\{ \text{"Gilmore"}, \text{"Hockey Night"} \}|} = 1/2.$$

Definition 9 completes our goal of providing a definition of confidence and support for general entity-relationship queries.

4.2 Comparison With Other Rule Languages

This section gives a brief comparison of our rule language and frequency definition to related rule languages. It is easy to see that the classic association rule approach based on frequent itemsets is a special case. For example, suppose we have two entity tables: `Transactions(number)` that stores transactions, and `Item(name)` for items, and a relational table `TransItems(TransNumber, ItemName)` that indicates which items appear in which transactions. Then for a given item, say “cola”, the query $Transactions(X) \wedge TransItems(X, \text{“cola”})$ returns the set of transactions involving “cola”, and the frequency of this query is the frequency of these transactions among all transactions.

Antonie and Zăiane [1] extend itemset rules with negations, and survey a number of search algorithms for finding frequent itemsets with negative conditions. Their search procedure is based on correlation analysis.

The WARMR system [4] considers queries that are conjunctions of literals (e.g., $P(X, Y)$). The user specifies a target table T ; the free query variables in a WARMR query are then bound to the key fields of T . If `WeekdayTV` is our target table, we would have two free query variables P for program and SN for station. All other variables are implicitly existentially quantified. For example, if `Customer` is the target table, the WARMR formula $Customer(A) \wedge Child(A, C) \wedge Buys(C, \text{“cola”})$ translates into the domain relational calculus as $\exists C. Customer(A) \wedge Child(A, C) \wedge Buys(C, \text{“cola”})$. If we assume that one of the conjuncts in a WARMR clause corresponds to the target table (e.g., $Customer(A)$), and all other appearances of the query variables are related to the target table by foreign key constraints (e.g., the first field in the `Child` table is a foreign key to the `Customer` table), then the reference domain as we have defined it is exactly the target table, and the frequency that WARMR assigns to a conjunction agrees with our definition. In this sense our definition of support for ER rules generalizes that for WARMR rules.

5 The Probability Axioms and A Priori Property

In order to ensure that Definition 7 yields well-defined probabilities, we verify three facts: (1) the frequency as defined never involves division by 0, so the frequency is well-defined. (2) The definition entails that frequencies are between 0 and 1 (inclusively). (3) The frequency of two mutually exclusive queries is the sum of their respective frequencies. This third property holds only with certain qualifications due to the restrictions on safe queries. The usual probability axioms include the requirement that (4) the probability of the whole space, or the “certain event” is 1. We discuss the extent to which this property holds for our definition of frequency. Finally, we show the APRIORI property: frequencies of conjunctions decrease monotonically, which is important for lattice search methods.

For the first fact, we have the following result. The notion of a valid ER query was specified in Definition 6.

Proposition 10 *Let F be a valid ER query whose free variables are X_1, \dots, X_m . Let \mathcal{D} be any database instance (without empty tables). Then $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) \neq \emptyset$.*

Proof. If F is valid, then for every maximal conjunction L of literals that occurs in F , we have $dom_{\mathcal{D}}(L, \{X_1, \dots, X_m\}) \neq \emptyset$. Since the reference domains of more complex formulas are the union of the domains of their subformulas, it follows that $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) \neq \emptyset$. ■

The next proposition guarantees that the ratios assigned by Definition 7 are properly bounded between 0 and 1.

Proposition 11 *Let F be an ER query in which the variables X_1, \dots, X_m are free such that F is valid for these variables. Let \mathcal{D} be a database instance. Then $\pi_{\langle X_1, \dots, X_m \rangle} tuples_{\mathcal{D}}(F) \subseteq dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\})$, where π is the projection operation of relational algebra.*

In the case in which X_1, \dots, X_m are exactly the free variables of F , we have $\pi_{\langle X_1, \dots, X_m \rangle} tuples_{\mathcal{D}}(F) = tuples_{\mathcal{D}}(F)$, so the proposition implies that the ratio $\frac{|tuples_{\mathcal{D}}(F)|}{|dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\})|}$ is between 0 and 1.

Proof. The proof is by induction on the structure of ER formula F . We begin by noting two basic facts about valid formulas, which follow easily from Definitions 2, 6, and 7.

1. If $C = C_1 \wedge \dots \wedge C_k$ is a maximal conjunction in F , then C contains a conjunction $X_1 = c_1 \dots \wedge X_m = c_m$ or a conjunct C_i that is a valid ER query.
 2. If $F_1 \vee F_2$ is a disjunction in F , then both of the disjuncts are valid ER queries.
- If F is an atomic formula of the form $P(t_1, \dots, t_k)$, then since F is valid for X_1, \dots, X_m , we have $\text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F)$.
 - Let F be a single atomic comparison of the form $Y \theta t$ where t is either a variable or a constant. Since F is valid, it must be of the form $X_1 = c$ where $m = 1$ (i.e., we just have a single free variable X_1 and the atomic formula requires X_1 to be equal to a constant c). So $\text{dom}_{\mathcal{D}}(F, \{X_1\}) = \{c\}$, and clearly $\pi_{X_1} \text{tuples}_{\mathcal{D}}(F) \subseteq \{c\}$.
 - Let F be a maximal conjunction of $k > 1$ formulas, such that $F = C_1 \wedge \dots \wedge C_k$.
 1. If F is a conjunction of the form $C \wedge X_1 = c_1 \dots \wedge X_m = c_m$, then $\text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \text{dom}_{\mathcal{D}}(C, \{X_1, \dots, X_m\}) \cup \{\langle c_1, \dots, c_m \rangle\}$. Clearly $\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F) \subseteq \{\langle c_1, \dots, c_m \rangle\}$, which is a subset of $\text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})$.
 2. Otherwise $\text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \bigcup_{i=1}^k \text{dom}_{\mathcal{D}}(C_i, \{X_1, \dots, X_m\})$. Since F is valid, by Observation 1 at least one of the conjuncts C_i is valid. So by inductive hypothesis,

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(C_i) \subseteq \text{dom}_{\mathcal{D}}(\{C_i, \{X_1, \dots, X_m\}\}).$$

Now since F is a conjunction involving C_i , it follows that

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F) \subseteq \pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(C_i)$$

and that

$$\text{dom}_{\mathcal{D}}(C_i, \{X_1, \dots, X_m\}) \subseteq \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\}),$$

which establishes the inductive hypothesis for this case.

- If F is $F_1 \vee F_2$, then by Clause 2 of the definition of a safe query, both F_1 and F_2 are valid and contain all the variables $\{X_1, \dots, X_m\}$ as free variables. So

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F) = \pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F_1) \cup \pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F_2).$$

Also, by inductive hypothesis,

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F_1) \subseteq \text{dom}_{\mathcal{D}}(F_1, \{X_1, \dots, X_m\})$$

and

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F_2) \subseteq \text{dom}_{\mathcal{D}}(F_2, \{X_1, \dots, X_m\}),$$

and by definition

$$\text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \text{dom}_{\mathcal{D}}(F_1, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F_2, \{X_1, \dots, X_m\}).$$

So

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F) \subseteq \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})$$

as required.

- If F is $\neg G$ for some formula G , then F is not a safe query, hence not an ER query, and the claim holds vacuously.

- If F is $\exists Y.G$, then $Y \notin \{X_1, \dots, X_m\}$, since the variables X_1, \dots, X_m are free in F . So

$$\text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\}) = \text{dom}_{\mathcal{D}}(G, \{X_1, \dots, X_m\}),$$

and

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(F) = \pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(G)$$

by the semantics of the existential quantifier. Clearly if F is valid, then so is G , so by inductive hypothesis

$$\pi_{\langle X_1, \dots, X_m \rangle} \text{tuples}_{\mathcal{D}}(G) \subseteq \text{dom}_{\mathcal{D}}(G, \{X_1, \dots, X_m\})$$

which completes the inductive proof.

■

The third fundamental property of probabilities is *finite additivity*, that the frequency of two mutually exclusive events is the sum of the individual frequencies. The difficulty with this property is not that it fails for our frequency definition, but that it is not straightforwardly expressed in our language of safe queries. For example, a natural formulation of finite additivity would be to require that $\text{fr}_{\mathcal{D}}(F) + \text{fr}_{\mathcal{D}}(\neg F) = \text{fr}_{\mathcal{D}}(F \vee \neg F)$. But if F is a safe query, then $\neg F$ is not safe, so the frequency $\text{fr}_{\mathcal{D}}(\neg F)$ is not defined. Another way to see the difficulty is to note that in standard probability theory (with a Boolean algebra of events), finite additivity is equivalent to the requirement that $\text{Pr}(A) = 1 - \text{Pr}(\bar{A})$, where \bar{A} is the complement of event A . But this cannot be expressed as a requirement on safe queries since the negation of a safe query is not itself safe.

However, we can show a qualified version of finite additivity. If S and F are valid safe queries with the same free variables, then the formulas $S \wedge F$ and $S \wedge \neg F$ are also valid safe queries. For these formulas we can show the following result.

Proposition 12 *Let S and F be valid safe queries with the same free variables $\{X_1, \dots, X_m\}$. Then for any database instance \mathcal{D} we have*

$$\text{fr}_{\mathcal{D}}([S \wedge F] \vee [S \wedge \neg F]) = \text{fr}_{\mathcal{D}}(S \wedge F) + \text{fr}_{\mathcal{D}}(S \wedge \neg F) = \frac{\text{tuples}_{\mathcal{D}}(S)}{\text{dom}_{\mathcal{D}}(S, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})}.$$

Proof. This follows from the definitions: We have $\text{dom}_{\mathcal{D}}([S \wedge F] \vee [S \wedge \neg F], \{X_1, \dots, X_m\}) = \text{dom}_{\mathcal{D}}([S \wedge F], \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}([S \wedge \neg F], \{X_1, \dots, X_m\})$, and since $\text{dom}_{\mathcal{D}}([S \wedge F], \{X_1, \dots, X_m\}) = \text{dom}_{\mathcal{D}}([S \wedge \neg F], \{X_1, \dots, X_m\}) = \text{dom}_{\mathcal{D}}(S, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})$, it follows that

$$\text{dom}_{\mathcal{D}}([S \wedge F] \vee [S \wedge \neg F], \{X_1, \dots, X_m\}) = \text{dom}_{\mathcal{D}}(S, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\}).$$

Clearly $\text{tuples}_{\mathcal{D}}([S \wedge F] \vee [S \wedge \neg F]) = \text{tuples}_{\mathcal{D}}(S)$, so

$$\text{fr}_{\mathcal{D}}([S \wedge F] \vee [S \wedge \neg F]) = \frac{\text{tuples}_{\mathcal{D}}(S)}{\text{dom}_{\mathcal{D}}(S, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})}.$$

Also, $\text{fr}_{\mathcal{D}}(S \wedge F) = \frac{\text{tuples}_{\mathcal{D}}(S \wedge F)}{\text{dom}_{\mathcal{D}}(S, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})}$ and $\text{fr}_{\mathcal{D}}(S \wedge \neg F) = \frac{\text{tuples}_{\mathcal{D}}(S \wedge \neg F)}{\text{dom}_{\mathcal{D}}(S, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})}$,
so

$$\text{fr}_{\mathcal{D}}(S \wedge F) + \text{fr}_{\mathcal{D}}(S \wedge \neg F) = \frac{\text{tuples}_{\mathcal{D}}(S)}{\text{dom}_{\mathcal{D}}(S, \{X_1, \dots, X_m\}) \cup \text{dom}_{\mathcal{D}}(F, \{X_1, \dots, X_m\})},$$

which was to be shown. ■

This result illustrates that two logically equivalent queries can have different frequencies in a given database instance, although their result tuples are always the same. In particular, although the queries $[S \wedge F] \vee [S \wedge \neg F]$ and S are logically equivalent, they have different reference domains: the domain of $[S \wedge F] \vee [S \wedge \neg F]$ includes also the domain of the query F . This is due to our closed-world assumption: since

the entities in the query F are among those mentioned in the query $[S \wedge F] \vee [S \wedge \neg F]$, they are included among the *potential* answers to the query, although in fact no entity satisfying F will be an actual answer to the query unless it is also an entity satisfying S .

The final standard property of probability measures on a Boolean algebra is that $P(X) = 1$, where X is the “certain event” that contains all possible outcomes. One difficulty with this property from the point of view of our frequency definition is again not so much that the property fails to hold but that it is not straightforward to express. A natural way to translate the axiom into a logical framework is to require that all tautologies or logically necessary queries receive probability 1. For example the query $Student(A) \vee \neg Student(A)$ is a tautology when viewed as a logical formula, but it is not a safe query. Another conceptually illuminating difficulty is that in our frequency definition, there is no *single* fixed space of possible outcomes or events that is independent of the query being asked. Rather, we define a space of possible outcomes dynamically for every query (i.e., $dom_{\mathcal{D}}(F, \{X_1, \dots, X_m\})$ for query F). For a *given* reference domain, the probability 1 property holds to the extent that we can express it. For example, if the only two possible genders are *male* and *female*, then the query $[Student(A) \wedge Gender(A, male)] \vee [Student(A) \wedge Gender(A, female)]$ receives frequency 1 in every database instance.

Finally we show that frequency as defined decreases monotonically with respect to conjunctions. This is important because many algorithms that search for frequent query formulas use this property to avoid exhaustive search. The following result guarantees that the frequency of a conjunction is less than the frequency of its conjuncts, which we refer to as the APRIORI property.

Proposition 13 (The Apriori Property) *Let \mathcal{D} be a database instance with valid ER query F_1 whose free variables are X_1, \dots, X_m and suppose that $F_1 \wedge F_2$ is also a valid ER query whose free variables are X_1, \dots, X_m . Then $fr_{\mathcal{D}}(F_1 \wedge F_2) \leq fr_{\mathcal{D}}(F_1)$.*

Proof. Clearly

$$tuples_{\mathcal{D}}(F_1 \wedge F_2) \subseteq tuples_{\mathcal{D}}(F_1),$$

and

$$dom_{\mathcal{D}}(F_1, \{X_1, \dots, X_m\}) \subseteq dom_{\mathcal{D}}(F_1 \wedge F_2, \{X_1, \dots, X_m\}).$$

So

$$\frac{|tuples_{\mathcal{D}}(F_1 \wedge F_2)|}{|dom_{\mathcal{D}}(F_1 \wedge F_2, \{X_1, \dots, X_m\})|} \leq \frac{|tuples_{\mathcal{D}}(F_1)|}{|dom_{\mathcal{D}}(F_1, \{X_1, \dots, X_m\})|}.$$

■

Discussion. Previous approaches to mining multi-relational rules such as WARMR mine rules for just one target table. Our approach in contrast can potentially search the entire space of queries for a given language bias, since by the proposition just established, the a priori property holds for the entire query space, not just for a fixed target table or key atom, given our definition of frequency and support. So compared to an iterative approach where we repeatedly apply a single-table rule miner to different tables in the database, our approach offers computational advantages. Intuitively, our approach combines the results of rule mining for separate tables when it considers rules that involve the separate tables at the same time. For example, suppose that for the *Student* table, we find that the query $Student(X) \wedge Age(X, 30)$ is infrequent. Then from Proposition 13 we can conclude that the query $Student(X) \wedge Age(X, 30) \wedge Professor(X)$ is infrequent as well. A traditional single-table rule mining system applied to both target tables would have to evaluate this conjunction twice, once with the target table *Student* and the second time with the target table *Professor*.

The price for the computational advantage of the a priori property holding throughout the query space is that our approach restricts the set of interesting queries compared to an iterative application of single-table rule mining. For example, it may be the case that the rule $Professor(X) \wedge Student(X) \rightarrow Age(X, 30)$ receives enough support if evaluated with respect to Professors (because it may be the case that most professors who are also taking courses as students are younger), but does not receive enough support if evaluated with respect to Students (perhaps because very few students are also professors to begin with). Our definition of support based on taking the union of the database tables can be seen as a *cautious* approach because if a query is frequent with respect to the union of two tables, it is frequent with respect to either

table. So a query that is frequent with respect to the union of the Professor and Student tables is frequent with respect to both.

6 Conclusion

The goal of this report was to extend the concept of confidence and support for a new class of association rules which we call entity-relationship rules. Entity-relationship rules are based on the domain relational calculus; they are much more flexible and expressive than standard itemset rules. ER rules allow for negation, nested Boolean combinations, and quantification. The main conceptual contribution of this report is a definition of frequency for entity-relationship queries. Instead of beginning with a specified target table or “key atom”, we dynamically define a reference or base domain of individuals for each ER query. The key idea of our definition is to take the base set of entities of a conjunctive query to be the union of the conjuncts’ base sets. For example, the frequency of the query $Professor(X) \wedge Customer(X)$ is computed with respect to the union of Professors and Customers. We proved that our frequency definition satisfies standard axioms for probabilities and validates the APRIORI property: the frequency of a conjunction is no greater than the frequency of any conjunct.

As usual in data mining, there is a tradeoff between the expressiveness of the rule or pattern language, and the difficulty of searching for significant patterns. Our rule language is very general and in practice a computational search for interesting entity-relationship rules will require a language restriction (bias). A central topic for future research is to explore language restrictions that make feasible a computational search for interesting entity-relationship rules.

Acknowledgements

This research was supported by Discovery Grants to the first and third author from the Natural Sciences and Engineering Council of Canada.

References

- [1] “Mining Positive and Negative Association Rules: An Approach for Confined Rules”, Maria-Luiza Antonie and Osmar R. Zaïane (2004). *8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 04)*, Springer Verlag LNCS 3202, pp 27-38, Pisa, Italy, September 20-24.
- [2] “Relational Completeness of data base sub-languages”, E. Codd (1972). In R. Rustin, editor, *Data Base Systems*, Prentice Hall.
- [3] “Attribute-value learning versus inductive logic programming: The missing links (extended abstract)”. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 1–8. Springer, Berlin 1998.
- [4] “Discovery of Relational Association Rules”, Luc Deshape and Hannu Toivonen (2001), Ch.8, in *Relational Data Mining*, eds. Saso Dzeroski and Nada Lavrac, Springer Berlin.
- [5] “Propositionalization Approaches to Relational Data Mining”, Stefan Kramer, Nada Lavrač and Peter Flach (2001), Ch.8, in *Relational Data Mining*, eds. Saso Dzeroski and Nada Lavrac, Springer Berlin.
- [6] “Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions”, G. Özsoyoğlu, Z.M. Özsoyoğlu, and V. Matos (1987), *ACM Transactions on Database Systems*, Vol.12:4, pp.566–592.
- [7] *Artificial Intelligence: A Modern Approach*, S. Russell and P. Norvig, (1988). Prentice Hall.

- [8] *Principles of Database and Knowledge-Base Systems*, Jeffrey D. Ullman (1988), Computer Science Press, Rockville, Maryland.